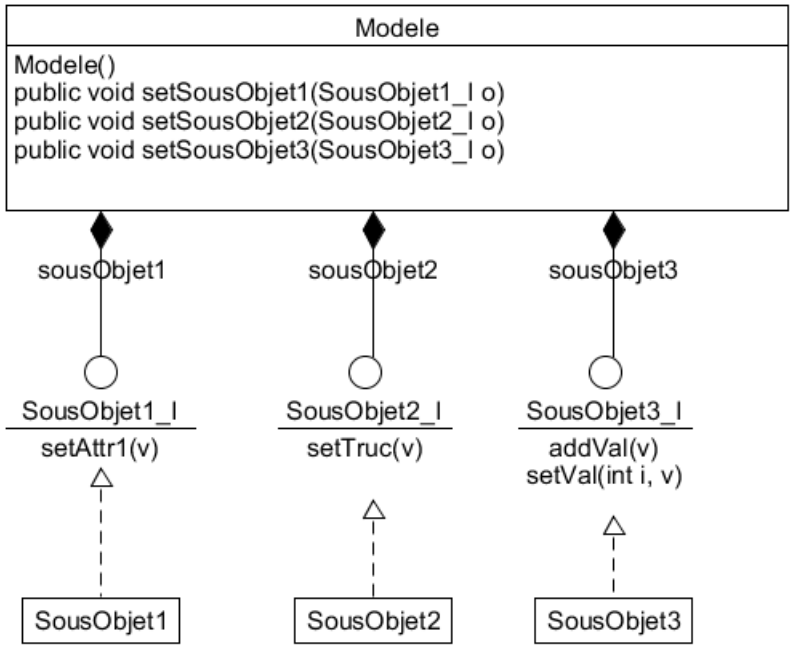
Exercice 08

Construire un modèle complexe

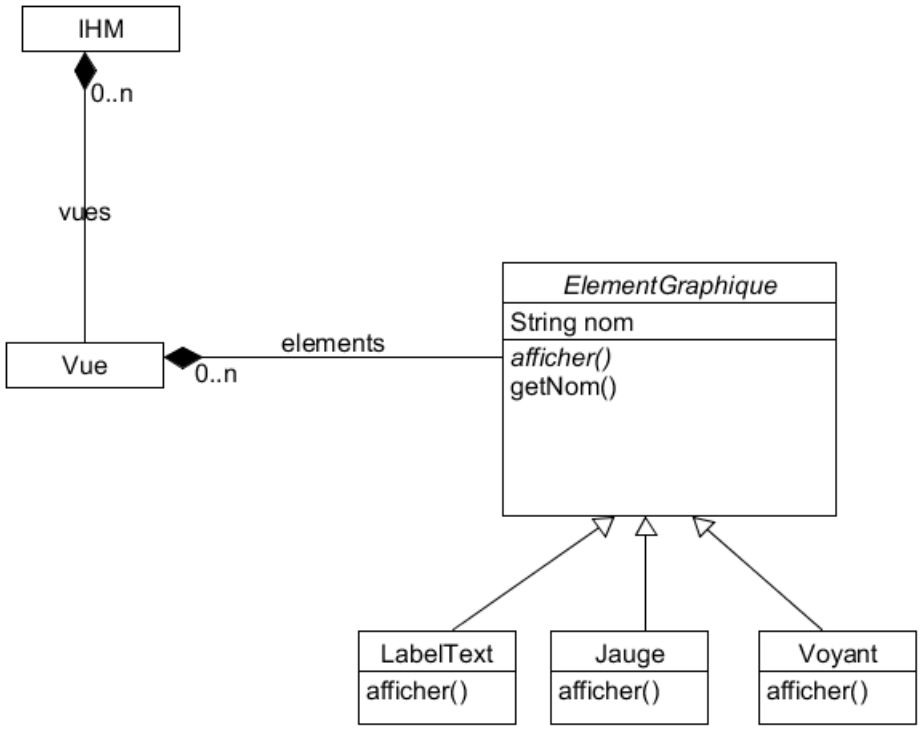
1. Enoncés

Soit un modèle d'informations (classe Modele) structurés en plusieurs sous-objets :



Le code de ce Modele ne doit pas être modifié.

Soit une IHM composées de plusieurs vues diverses et variées.
Chaque vue est composée de plusieurs éléments graphiques.
Un élément graphique peut être de différents types (texte, voyant, jauge, barre de progression, …).



Ce code d'IHM peut être enrichi.

Le modèle est utilisé par une application qui réalise des traitements.
Ces traitements sont susceptibles de modifier certains attributs des sous-objets à travers les méthodes de ces sous-objets (setteurs, addeurs, maj, …).

On veut faire en sorte que les éléments graphiques soient notifiés lorsque l'application change des attributs de ces sous-objets.

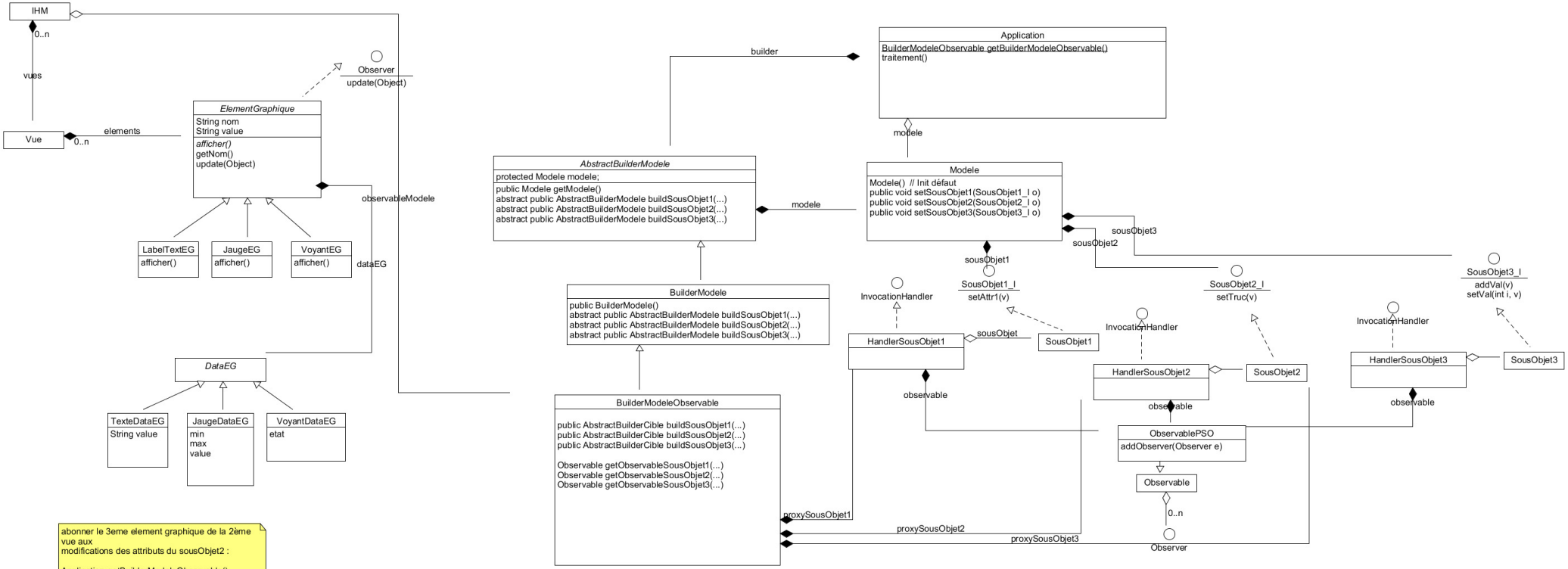
C'est l'IHM qui décide quel élément graphique est notifié par tel sous-objet.
Si c'est le cas, on part du principe qu'un élément graphique est systématiquement notifié quand au moins un des attributs d'un même sous-objet est modifié.
La donnée notifiée est un objet que l'élément graphique utilise pour rafraichir l'élément graphique.

Plusieurs éléments graphiques peuvent être notifiés par plusieurs sous-objets différents.

Un sous-objet peut ne pas faire de notification. C'est lors de l'instanciation du modèle que l'on décide quels sous-objets sont susceptible de notifier.

On veut que la solution de conception soit la plus propre possible et donc soit souple et systématique dans son implémentation (ce qui est le propre des Designs Patterns).

2. Correction



abonner le 3eme element graphique de la 2ème vue aux modifications des attributs du sousObjet2 :

```
Application.getBuilderModeleObservable()
  .getObservableSousObjet2("Jauge")
    .addObserver(vues[2].elements[3])
```

ElementGraphique :

```
update(Object o):
   dataEG=o;
   afficher();
```

```
public AbstractBuilderCible buildSousObjet1(...)
{
  super.buildSousObjet1(...);
  proxySousObjet1 =
    Proxy.newProxyInstance(Application.class.getClassLoader(),
                           new Class<?>[] { SousObjet1_I.class },
                           new HandlerSousObjet1(
                                 getModele().getSousObjet1()));
  getModele.setSousObjet1(proxySousObjet1);
}
```

class HandlerSousObjet1 :

```
invoke:
  observable.notifier(new JaugeDataEG(...))
```

class HandlerSousObjet2 :

```
invoke:
  observable.notifier(new TexteDataEG(...))
```

class HandlerSousObjet3 :

```
invoke:
  observable.notifier(new VoyantDataEG(...))
```